JOURNAL
COMPUTER SCIENCES INSTITUTE

JCSI 27 (2023) 171–177
Received: 13 April 2023
Accepted: 15 May 2023

Comparative analysis of data reading performance from the Salesforce platform using GraphQL, REST and SOAP interfaces

Analiza porównawcza wydajności odczytu danych z platformy Salesforce przy wykorzystaniu interfejsów GraphQL, REST oraz SOAP

Ryszard Roman Rogalski*

Department of Computer Science, Lublin University of Technology, Nadbystrzycka 36B, 20-618 Lublin, Poland

Abstract
The article describes a comparative analysis of data reading from tables in the Salesforce environment using three different application programming interfaces. The popularity of the Salesforce platform and the release of the GraphQL interface on it on October 5, 2022 were an inspiration to perform the research. Based on the literature reviewed, there was no such study for the Salesforce platform. The performance of reading data from the Salesforce platform was investigated using an automation script. For four tables containing a different number of rows, 8 types of queries were repeatedly executed using each of the three interfaces. It was found that depending on the number of rows, either REST API or SOAP API should be considered. In all cases, the lowest performance was observed while using GraphQL API.

Keywords: Salesforce; performance; GraphQL; API

Streszczenie
Artykuł opisuje analizę porównawczą odczytu danych z tabel w środowisku Salesforce przy pomocy trzech różnych interfejsów programowania aplikacji. Popularność platformy Salesforce oraz udostępnienie na niej interfejsu GraphQL dnia 5 października 2022 roku były inspiracją do przeprowadzenia badań. Na podstawie przeanalizowanej literatury stwierdzono brak takiego badania dla platformy Salesforce. Zbadano wydajność odczytu danych z platformy Salesforce przy pomocy skryptu automatyzującego. Dla czterech tabel zawierających inną liczbę wierszy wielokrotnie wykonano 8 rodzajów zapytań przy pomocy każdego z trzech interfejsów. Stwierdzono, że zależnie od liczby wierszy należy rozważyć użycie REST API lub SOAP API. We wszystkich przypadkach najniższą wydajność zaobserwowano podczas zastosowania GraphQL API.

Słowa kluczowe: Salesforce; wydajność; GraphQL; API

*Corresponding author
Email address: ryszard.rogalski@pollub.edu.pl (R. Rogalski)

1. Wstęp

Branża informatyczna wykorzystuje tradycyjne relacyjne bazy danych od około 40 lat. Jednak w ostatnich latach nastąpiła znaczna przemiana branży IT w zakresie aplikacji komercyjnych. Samodzielne aplikacje zostały zastąpione rozwiązaniami serwerowymi. Niższe opłaty, elastyczność oraz model naliczania kosztu usługi dopiero po jej wykonaniu to główne przyczyny, które spowodowały, że obliczenia rozproszone stały się rzeczywistością [1].

Database-as-a-Service (DBaaS) jest usługą, która zapewnia bazę danych lub dedykowaną instancję do zarządzania danymi w chmurze, bez ponoszenia kosztów czasowych związanych z zarządzaniem infrastrukturą, udostępnianiem sprzętu, konfiguracją silnika, aktualizacjami i kopiami zapasowymi. Baza danych jest hostowana w zdalnym centrum danych i może być współdzielona przez użytkowników w przejrzysty sposób. Usługa taka obejmuje na przykład tworzenie tabel lub ładowanie i dostęp do danych w tabelach. Aby zaoferować swobodny dostęp do bazy danych, tego typu rozwiązania często dostarczają interfejsy programistyczne i wykonują operacje bazodanowe za ich pośrednictwem [2].

Platforma Salesforce dostarcza wiele interfejsów programistycznych do komunikacji z bazą danych [3], lecz zdecydowano się dokonać analizy wydajności trzech interfejsów, które są powszechnie używane w innych systemach. Dokonano analizy tej platformy ze względu na jej udział w rynku i rosnącą popularność [4]. W badaniu przeprowadzono testy wydajności zapytań odczytujących dane ze względu na to, że to jedyna dostępna obecnie operacja bazodanowa udostępniona w interfejsie GraphQL (*Graph Query Language*) na platformie Salesforce [5].

2. Interfejs programowania aplikacji GraphQL

GraphQL to propozycja alternatywnego języka zapytań i silnika wykonywania dla interfejsów programowania aplikacji internetowych, który ma na celu rozwiązanie problemów z dostępem do danych i wersjonowaniem interfejsów [6]. Jest to język przeznaczony dla interfejsów wykorzystywanych głównie na stronach internetowych. Zaproponowany przez firmę Facebook w 2016 roku, stanowi alternatywę dla interfejsów takich

jak REST lub SOAP [7]. Oficjalna specyfikacja określa GraphQL jako język zapytań i silnik wykonawczy, który służy do opisu możliwości i wymagań modelu danych dla aplikacji klient-serwer [8].

3. Przegląd literatury

Uwzględniona literatura przedstawia analizę porównawczą trzech interfejsów programowania aplikacji: SOAP, REST oraz GraphQL. Nie przedstawia ona jednak analizy tych interfejsów na platformie Salesforce. Autorzy artykułu [7] przeprowadzili badanie przy pomocy trzech języków programowania: C#, Java oraz PHP. W pracy niestety nie sprecyzowano rodzajów wykonywanych zapytań. Autorzy dla 50 wirtualnych użytkowników wykonujących zapytania jednocześnie uzyskali wyniki przedstawione w Tabeli 1.

Tabela 1: Średni czas odpowiedzi dla zapytania wykonanego przy użyciu danego języka oraz interfejsu [7]

Język	Interfejs	Średni czas odpowiedzi (s)
C#	SOAP	74000
Java	SOAP	127600
PHP	SOAP	107800
C#	REST	56800
Java	REST	107400
PHP	REST	102800
C#	GraphQL	32200
Java	GraphQL	54000
PHP	GraphQL	48800

Jednak wyniki świadczyły, że niezależnie od wykorzystanego języka programistycznego to interfejs GraphQL okazał się najszybszy.

Autorzy kolejnej pracy [9] podkreślają, że dyskusje na temat najbardziej optymalnego rozwiązania dotyczącego tworzenia usług sieciowych wystawiających API nie są rozstrzygnięte. Autorzy przeprowadzili test, w którym wykonano 500 zapytań w czasie 5 sekund, uzyskując średni czas wykonania jednego zapytania na poziomie 0,01 sekundy. Wyniki pokazały, że w przypadku użycia GraphQL, wraz ze wzrostem liczby kolejnych zapytań czas odpowiedzi znacznie się wydłużał.

1. REST okazał się wydajniejszy w przypadku pobierania wszystkich zasobów z bazy.
2. GraphQL był efektywniejszy w przypadku pobierania wyszczególnionych danych.

Erlandsson oraz Remes [10], również przeprowadzili porównanie wydajności tych trzech interfejsów. Postawili oni dwie hipotezy. Pierwsza zakładała, że interfejs GraphQL, będzie wydajniejszy tylko, gdy liczba wierszy w tabeli będzie dostatecznie duża oraz wybierane będą konkretne wiersze. Wyniki świadczyły o tym, że GraphQL wypada najgorzej, niezależnie od liczby wierszy w tabeli lub liczby wybieranych wierszy. Druga hipoteza zakładała, że to interfejs SOAP będzie osiągał najgorsze wyniki, niezależnie od przypadku testowego. Okazało się ponownie, że to GraphQL był najmniej wydajny podczas prowadzonych testów.

Każda praca przedstawia zupełnie inne wyniki, co świadczy o tym, że zależnie od środowiska, na którym prowadzone są badania, można uzyskać odmienne rezultaty. Podczas analizy literatury nie znaleziono testów przeprowadzonych w środowisku Salesforce.

4. Metoda badawcza

4.1. Przygotowanie danych i wybór środowiska

Przed wykonaniem testów wydajnościowych przygotowano tabele w bazie danych. Utworzono cztery tabele zawierające identyczny model danych. Dla każdej z tabel zostało utworzone 8 kolumn o następujących typach:

1. Text (pole indeksowane, unikalne).
2. Number.
3. Currency.
4. Date.
5. Date/Time.
6. Checkbox
7. Picklist.
8. Multi-Select Picklist.

Dla każdej z czterech tabel utworzono następującą liczbę wierszy: 100, 1000, 10 000 oraz 500 000. Wartości kolumn w wierszach zostały wygenerowane pseudolosowo. Listing 1 przedstawia kod napisany w języku *Apex* odpowiedzialny za przypisywanie wartości kolumnom.

Listing 1: Kod do generowania danych psuedolosowych

```
/* pseudolosowa wartość True/False*/
Boolean__c = Math.random() < 0.5,
/* pseudolosowa data*/
Date__c = Date.newInstance(
              (Integer) Math.floor
               (Math.random() * 53) + 1970,
               (Integer) Math.floor
               (Math.random() * 12) + 1,
              (Integer) Math.floor
               (Math.random() * 31) + 1
),
/* pseudolosowa wartość Currency*/
Currency__c = (Integer)  Math.floor(
       Math.random() * 100000) + 1 + 0.5,
/* pseudolosowa wartość Number*/
Number__c = (Integer) Math.floor(
       Math.random() * 99999) + 1
```

Ze względu na ograniczenia platformy Salesforce, kod który generował 500 000 wierszy dla ostatniej tabeli został umieszczony w klasie implementującej interfejs *Batchable* (Listing 2), umożliwiający operowanie na większej liczbie rekordów.

Listing 2: Implementacja interfejsu *Batchable* w Salesforce

```
public without sharing class
    BatchHelp implements Database.Batchable<SObject>{
/* klasa implementująca interfejs Batchable (Salesforce) */

public void execute(Database.BatchableContext info,
       List<Account> scope){
    /*metoda wykonawcza w interfejsie */

    String query = 'SELECT COUNT() FROM Table_500000__c';

    Integer records = Database.countQuery(query);
    /*sprawdzenie aktualnej liczby wierszy w tabeli */
}
```

Utworzona została również piąta tabela, w której utworzono kolumny zawierające referencje do poprzednich tabel. Utworzono wiersze w tej tabeli zachowując integralność referencyjną (Listing 3).

Listing 3: Tworzenie wierszy zachowując integralność referencyjną

```
List<Relationship_Table__c> relation =
        new List<Relationship_Table__c>();
/*zapewnienie integralności referencyjnej w nowych wierszach*/
for(Table_500000__c x : insertsDML){
    relation.add(new Relationship_Table__c(
            Table_500000__c = x.Id
    ));
insert relation;
/*wykonanie operacji DML dodania wierszy do tabeli*/
```

Podczas badania wykorzystano odpowiednią wersję platformy Salesforce, umożliwiającą przechowywanie takiej ilości danych. Przykładowo wersja deweloperska oferuje tylko 5 megabajtów pamięci do przechowywania danych. W Tabeli 2 przedstawiono parametry użytej instancji Salesforce.

Tabela 2: Parametry wykorzystanej instancji

Parametr	Wartość
Edycja	Enterprise
Instancja	EU46
Lokalizacja	Frankfurt, Niemcy
Wersja	Spring 23

4.2. Przebieg badania

Badanie zostało przeprowadzone przy pomocy skryptu automatyzującego. Skrypt został napisany w języku *Python* (wersja 3.9.13). Podczas zliczania rezultatów upewniono się, że jedyną wartością mierzoną jest czas odpowiedzi serwera na żądanie oraz czas przetworzenia zapytania POST, a przypisania zmiennych oraz wszelkie inne instrukcje zostały wyniesione poza obszar kodu, w którym dokonywano pomiaru (Listing 4). Zdecydowano się na symulację komunikacji serwera zewnętrznego z platformą Salesforce i wysyłano jedno żądanie jednocześnie.

Listing 4: Pomiar wykonania zapytania przy pomocy protokołu HTTP

```
start_time = time.perf_counter()
#rozpoczecie pomiaru czasu
response = requests.post(instance_url,
                         headers=headers,
                         json=json_variable)
#wykonanie zapytania HTTP POST
end_time = time.perf_counter()
#zakonczenie pomiaru czasu
```

Konfiguracja sprzętowa komputera, na którym został uruchomiony skrypt została przedstawiona w Tabeli 3.

Tabela 3: Parametry sprzętowe komputera

Parametr	Wartość
Pamięć RAM	16 GB
Procesor	Apple M1 (8 rdzeni)
Dysk	SSD 256 GB
Częstotliwość zegara	2064 - 3220 MHz
Architektura	ARM
System operacyjny	macOS Ventura 13.2.1

W ramach eksperymentu zrealizowano następujące scenariusze badawcze:

1. Zmierzenie czasu odpowiedzi serwera na żądanie wierszy z wyszczególnieniem wszystkich kolumn.
2. Zmierzenie czasu odpowiedzi serwera na żądanie wierszy z wyszczególnieniem tylko identyfikatora wiersza.
3. Zmierzenie czasu odpowiedzi serwera na żądanie konkretnego wiersza opisanego przy pomocy kolumny nieindeksowanej.
4. Zmierzenie czasu odpowiedzi serwera na żądanie konkretnego wiersza opisanego przy pomocy kolumny indeksowanej.
5. Zmierzenie czasu odpowiedzi serwera na żądanie wierszy z zastosowaniem sortowania według kolumny nieindeksowanej.
6. Zmierzenie czasu odpowiedzi serwera na żądanie wierszy z zastosowaniem sortowania według kolumny indeksowanej.
7. Zmierzenie czasu odpowiedzi serwera na żądanie wierszy z wartością w kolumnie typu *Currency* większą niż zadana.
8. Zmierzenie czasu odpowiedzi serwera na żądanie wierszy z zastosowaniem podzapytania w klauzuli *WHERE*.

W scenariuszach opisujących wybranie wierszy, dla tabel o liczbie wierszy większej niż 200 wykonano serie zapytań, zwracających po 200 wierszy każde. Odpowiedzi interfejsów na platformie Salesforce w takich przypadkach zwracają adresy dla których powinno wykonane zostać kolejne zapytanie zwracające kolejne 200 wierszy (Listing 5). W ten sam sposób działa zarówno SOAP, REST oraz GraphQL API (na platformie Salesforce). Ze względu na ograniczenia występujące na platformie Salesforce, dla tabel zawierających 10 000 oraz 500 000 wierszy wybrano ich jedynie 4000. Dla pozostałych tabel wybrano wszystkie wiersze. Ograniczenie do 4000 wierszy wynika ze sposobu implementacji interfejsu GraphQL na platformie Salesforce, czyli limitem maksymalnej wartości (2000) opcjonalnych klauzul *LIMIT* oraz *OFFSET* [10]. Operacje wysłania opisanych żądań HTTP powtórzono 19 razy dla trzech interfejsów oraz czterech rozmiarów tabel, zatem przeprowadzono 228 prób.

Listing 5: Uzyskanie adresu kursora ze zwróconej odpowiedzi

```
while hasNextPage:
    if response.status_code == 200:
        #sprawdzenie statusu odpowiedzi serwera

        data = response.json()['data']['uiapi']
        ['query'][TABLE_NAME]
        #transformacja zwroconego tekstu do postaci JSON

        totalCount = data['totalCount']
        pageInfo = data['pageInfo']
        cursor = data['pageInfo']['endCursor']
        #uzyksanie informacji o zwroconym kursorze,

        hasNextPage = pageInfo['hasNextPage']
```

W Tabeli 4 przedstawiono zapytania przekazywane w żądaniu podczas realizacji scenariuszy badawczych.

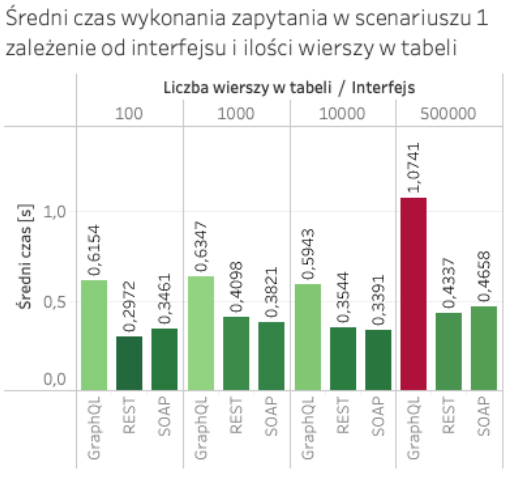
Zapytania w tabeli przedstawione napisane są języku SOQL (*Salesforce Object Query Language*). SOQL jest podobny do instrukcji SELECT w powszechnie używanym języku SQL, ale został zaprojektowany specjalnie dla danych Salesforce.

Tabela 4: Zapytania przekazywane w żądaniu w danym scenariuszu badawczym

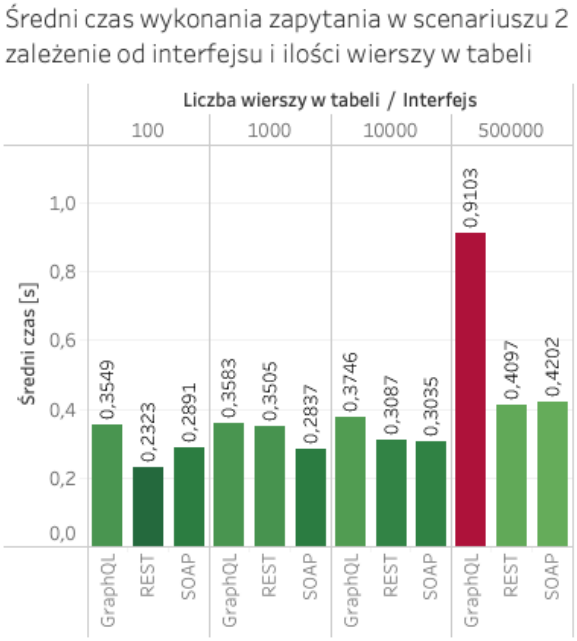
Scenariusz badawczy	Zapytanie SOQL
1	SELECT Id, Boolean__c, Currency__c, Date__c, Date_Time__c, Index__c, Multi_Pick_List__c, Number__c, Picklist__c FROM Table__c
2	SELECT Id FROM Table__c
3	SELECT Id FROM Table__c WHERE Currency__c = 0.2022
4	SELECT Id FROM Table__c WHERE Index__c = '1'
5	SELECT Id, Boolean__c, Currency__c, Date__c, Date_Time__c, Index__c, Multi_Pick_List__c, Number__c, Picklist__c FROM Table__c ORDER By Currency__c ASC
6	SELECT Id, Boolean__c, Currency__c, Date__c, Date_Time__c, Index__c, Multi_Pick_List__c, Number__c, Picklist__c FROM Table__c ORDER By Index__c ASC
7	SELECT Id, Boolean__c, Currency__c, Date__c, Date_Time__c, Index__c, Multi_Pick_List__c, Number__c, Picklist__c FROM Table__c WHERE Currency__c > 10
8	SELECT Id, Boolean__c, Currency__c, Date__c, Date_Time__c, Index__c, Multi_Pick_List__c, Number__c, Picklist__c FROM Table__c WHERE Id IN (SELECT Table__c FROM Relationship_Table__c)

5. Wyniki

Rysunki 1–2 przedstawiają średni czas odpowiedzi serwera zależenie od użytego interfejsu i liczby wierszy w odpytywanej tabeli.

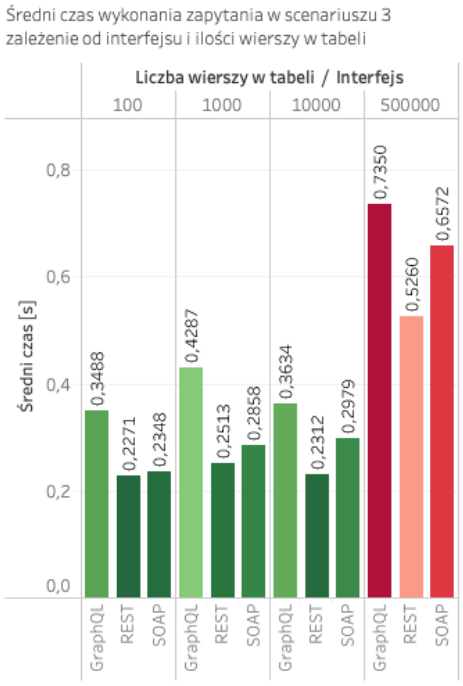


Rysunek 1: Średni czas odpowiedzi serwera w scenariuszu badawczym numer 1.

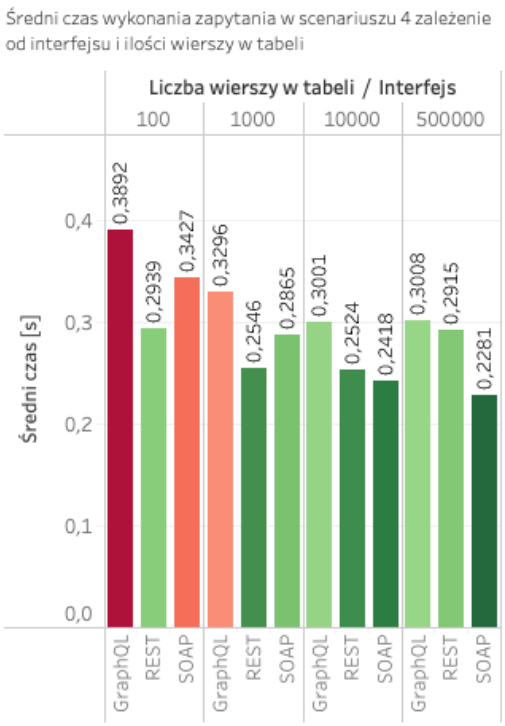


Rysunek 2: Średni czas odpowiedzi serwera w scenariuszu badawczym numer 2.

Otrzymane rezultaty świadczą o tym, że zależnie od liczby wierszy najwydajniejsze są interfejsy REST oraz SOAP, a najmniej wydajny GraphQL, natomiast wraz ze wzrostem liczby wierszy w tabelach interfejs GraphQL staje się coraz wolniejszy w porównaniu do pozostałych interfejsów. Rysunki 3–4 przedstawiają średni czas odpowiedzi serwera zależenie od użytego interfejsu i liczby wierszy w odpytywanej tabeli, lecz dla jednego wybieranego wiersza. Wiersz wybierano przy pomocy kolumny indeksowanej oraz kolumny nieindeksowanej.

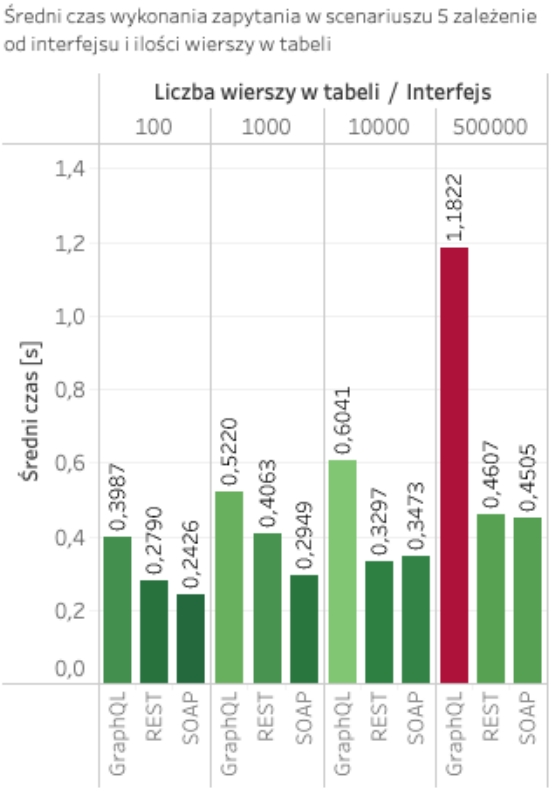


Rysunek 3: Średni czas odpowiedzi serwera w scenariuszu numer 3.

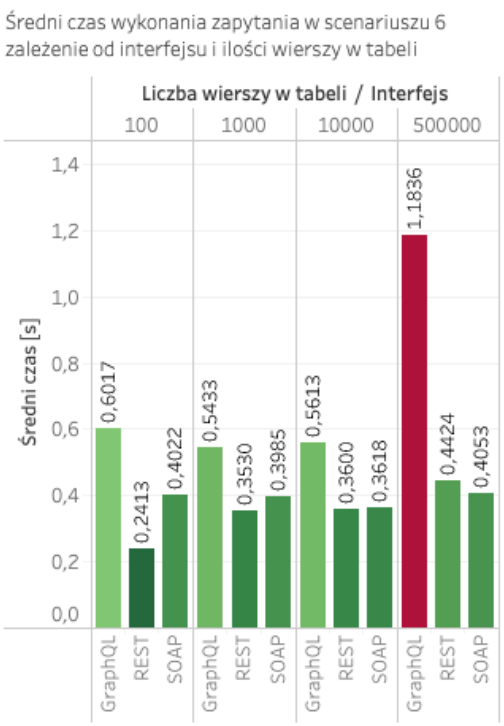


Rysunek 4: Średni czas odpowiedzi serwera w scenariuszu badawczym numer 4.

Rysunki 5–6 przedstawiają średni czas odpowiedzi serwera zależenie od użytego interfejsu i liczby wierszy w odpytywanej tabeli. Zapytania w tym wypadku zostały posortowane przy pomocy kolumny indeksowanej oraz kolumny nieindeksowanej.

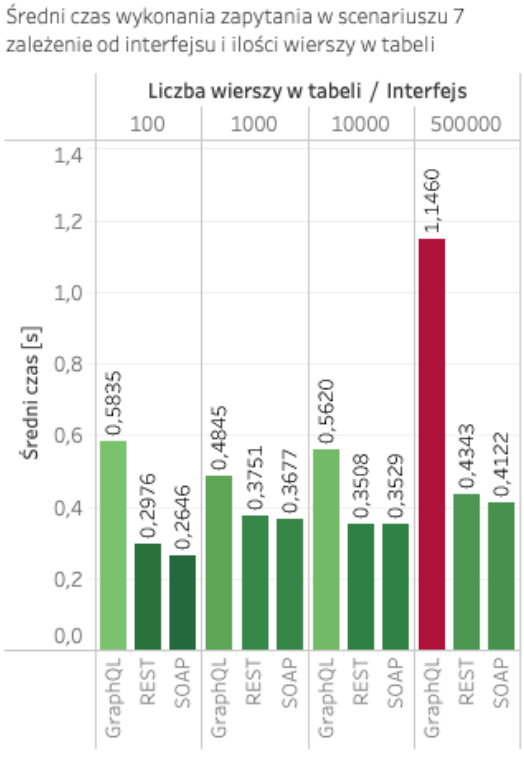


Rysunek 5: Średni czas odpowiedzi serwera w scenariuszu badawczym numer 5.

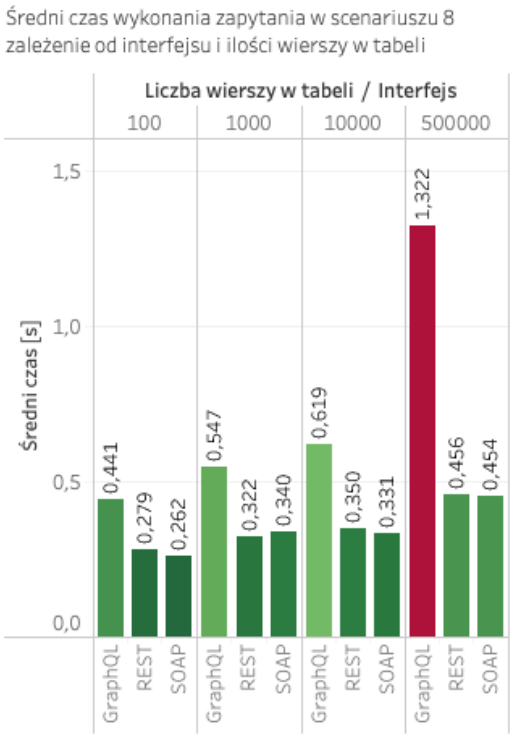


Rysunek 6: Średni czas odpowiedzi serwera w scenariuszu badawczym numer 6.

Rysunki 7–8 przedstawiają średni czas odpowiedzi serwera zależenie od użytego interfejsu i liczby wierszy w odpytywanej tabeli, lecz w zapytaniu zostało zastosowanie filtrowanie. Zastosowano filtrowanie po polu typu *Currency* oraz filtrowanie z użyciem przy podzapytania w klauzuli *WHERE*.

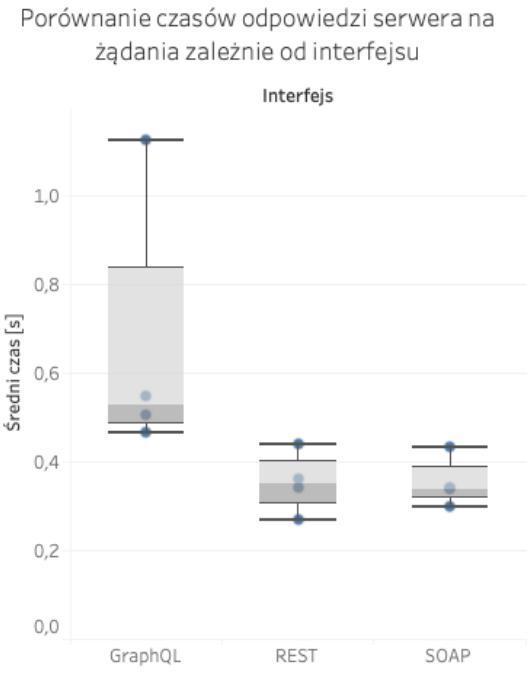


Rysunek 7: Średni czas odpowiedzi serwera w scenariuszu badawczym numer 7.



Rysunek 8: Średni czas odpowiedzi serwera w scenariuszu badawczym numer 8.

Rysunek 9 przedstawia porównanie czasów odpowiedzi serwera we wszystkich przeprowadzonych eksperymentach.



Rysunek 9: Porównanie średnich czasów odpowiedzi serwera we wszystkich scenariuszach badawczych.

W Tabeli 5 przedstawiono szczegółowe wyniki dotyczące zależności interfejsu oraz liczby wierszy w tabeli. Podsumowano wszystkie przeprowadzone scenariusze badawcze oraz przedstawiono średni czas odpowiedzi jak i odchylenie standardowe.

Tabela 5: Szczegółowe wyniki wszystkich przeprowadzonych scenariuszy

Interfejs	Liczba wierszy w tabeli	Średni czas odpowiedzi (s)	Odchylenie standardowe czasu odpowiedzi (s)
GraphQL	100	0,47	0,52
	1000	0,51	0,39
	10000	0,55	0,42
	500000	1,13	0,46
REST	100	0,27	0,16
	1000	0,36	0,38
	10000	0,34	0,37
	500000	0,44	0,56
SOAP	100	0,30	0,31
	1000	0,34	0,32
	10000	0,34	0,36
	500000	0,43	0,59

6. Analiza wyników i wnioski

Różnica pomiędzy średnimi wynikami uzyskanymi przy pomocy interfejsów REST oraz SOAP jest na tyle znikoma (biorąc pod uwagę czynniki takie jak obciążenie serwera oraz obciążenie sieci), że nie stwierdza się różnicy w wydajności działania dla większości przypadków. Wyjątkiem jest sytuacja, gdy liczba rekordów w tabeli jest istotnie niewielka (poniżej tysiąca) - wtedy średnio o 11% szybsze jest REST API.

Wyniki świadczą o tym, że niezależnie od wybranego scenariusza badawczego, a więc liczby oraz typu kolumn, filtrowania czy liczby wierszy sprecyzowanych w zapytaniu najmniej wydajne jest GraphQL API. Należy również zwrócić uwagę, że zwiększenie liczby wierszy w odpytywanej tabeli drastycznie wpływa na czas wykonywania zapytania dla interfejsu GraphQL. Pomiędzy liczbą wierszy 10 000, a 500 000 dla interfejsów REST oraz SOAP, zauważono średnią różnice ok. 29%, a dla GraphQL wynosi ona już 105%. Uzyskano średnie odchylenie standardowe interfejsu GraphQL większe o 17% niż dla interfejsów REST oraz SOAP, co oznacza że czas uzyskiwania odpowiedzi od bazy danych jest mniej przewidywalny podczas użycia GraphQL. Najstabilniejszym interfejsem programowania aplikacji jest REST, a wykorzystanie SOAP wygenerowało średnio o 7% większe odchylenie standardowe.

Na podstawie uzyskanych wyników można jednoznacznie stwierdzić, że z pośród trzech interfejsów programowania aplikacji udostępnionych na platformie Salesforce to GraphQL jest najmniej wydajny. Są to wyniki zupełnie przeciwne do tych uzyskanych przez autorów badania przeprowadzonego w roku 2020 [7]. Z drugiej strony wyniki pokrywają się jednoznacznie z wynikami jakie otrzymali Erlandsson oraz Remes [10].

Podsumowując, można stwierdzić, że wybór interfejsu, który zostanie użyty podczas wytwarzania oprogramowania powinien być determinowany środowiskiem, które dane interfejsy udostępnia. Nie można jednoznacznie stwierdzić, który interfejs jest wydajniejszy bez określenia platformy, bądź środowiska.

Literatura

[1] J. H. Bhatti, B. B. Rad, Databases in Cloud Computing: A literature review, International Journal of Information Technology and Computer Science 9(4) (2017) 9–17, https://doi.org/10.5815/ijitcs.2017.04.02.

[2] L. A. B. Silva, C. Costa, J. L. Oliveira, A common API for delivering services over multi-vendor cloud resources, Journal of Systems and Software 86(9) (2013) 2309–2317, https://doi.org/10.1016/j.jss.2013.04.037.

[3] Dokumentacja interfejsów programowania aplikacji w Salesforce, https://developer.salesforce.com/docs/apis , [04.04.2023].

[4] Ranking systemów CRM dla roku 2023, https://www.pcmag.com/picks/the-best-crm-software , [04.04.2023].

[5] Dokumentacja Salesforce dotycząca GraphQL, https://developer.salesforce.com/docs/platform/graphql/references/graphql?meta=Summary , [04.04.2023].

[6] A. Quiña-Mera, P. Fernandez, J. M. García, A. Ruiz-Cortés, GraphQL: A Systematic Map-ping Study, ACM Computing Surveys 55(10) (2023) 1–35, https://doi.org/10.1145/3561818.

[7] J. Sayago Heredia, E. Flores-García, A. R. Solano, Comparative analysis between standards oriented to web services: SOAP, REST and GRAPHQL, Proceedings of the Applied Technologies: First International Conference, ICAT 2019, Quito, Ecuador (2019) 286–300, https://doi.org/10.1007/978-3-030-42517-3_22.

[8] Oficjalna dokumentacja GraphQL, https://spec.graphql.org/June2018/ , [04.04.2023].

[9] P. Margański, B. Pańczyk, Analiza porównawcza technologii REST i GraphQL, Journal of Computer Sciences Institute 19 (2021) 89–94, https://doi.org/10.35784/jcsi.2473.

[10] P. Erlandsson, J. Remes, Performance Comparison between GraphQL, REST & SOAP, University of Skovde, Dissertation, https://www.diva-portal.org/smash/record.jsf?pid=diva2:1449837, 2020, [04.04.2023].